Seeing how users in this sub and interviewers oppose to grinding LeetCode, I have decided to write a guide to help those who need to grind LeetCode.
First of all, if you think studying CS fundamentals alone can land you offers, you may stop reading here. This guide is intended for those who would like to equip themselves with the necessary skills through LeetCode to tackle technical interviews. Grinding LeetCode is more than just memorizing answers, you have to learn the problem-solving patterns by heart and apply them to similar problems. The number of problems you have solved in LeetCode is only one of the indicators of your familiarness to the patterns, learning the patterns is more than only numbers.

Checkpoint 0: Beyond the CS Fundamentals

This guide assumes that you have at least heard of the basic tricks such as two-pointers and bit manipulation from CTCI or similar books. You do not have to master them, knowing what they are can help you study the solutions from LeetCode better. If you have studied only the CS fundamentals, you may want to have a quick look at the books before starting LeetCode.

Easy Problems

Easy problems are intended to help you get familiar with the basic tricks. Usually, they have trivial brute force solutions. What you need to learn is to apply the tricks to improve your brute force solutions.

Checkpoint 1: Practicing the Basic Tricks

If you randomly open a few easy problems of each data structure or algorithm and you can pinpoint the optimal solutions and implement them in a few minutes, you may move on to the next checkpoint.

Study Guide

1. Sort the problems by acceptance rate descending. Problems with higher acceptance rates are relatively easier among the pool of easy problems.
2. Try to solve the problems with no hints at least with brute force solutions.
3. It is tempting, but not helpful, to abuse the "run" button. Try Easy ones with a goal to get accepted on the first submission, since this more realistically models a whiteboard situation. It forces you to think of all the use cases yourself. Thanks /u/dylan_kun for the tip.
4. Study how the top solutions apply the tricks to improve the performance. Sometimes solutions are up-voted just because they are short and they may not be well documented. Read also the comments below and do not feel shame to ask for clarifications.
5. Once you are comfortable with the basic problem-solving patterns, go back to checkpoint 1 and decide if you would like to move on.

Medium Problems

Medium problems are intended to train your skills in seeing through the problems. They are usually disguises or variations of easy problems. Brute force solutions sometimes may lead to time limit exceeded (TLE). What you need to learn is identifying what solving patterns the problems are asking for.

Checkpoint 2: Problem Pattern Recognition

If you randomly open a few medium problems of each data structure or algorithm and you can identify what problems they are disguising at and can implement close-to-optimal solutions within half hour, you are ready to challenge the hard problems.

Study Guide

1. Carefully read each word of the problem statements and look for hints about solving patterns. For example, the number of ways for a task indicates DP, string transformation with dictionary indicates BFS / DFS / Trie, looking for duplicate or unique elements indicates hashing or bit manipulation, parsing indicates the use of stack. If you need a compiled list of tricks and indicators of when to use what, you may check out the book Competitive Programmer’s Handbook. Thanks /u/ShadowOfOrion for the tip.
2. When you have a rough idea about the direction, you are half way to go. Try to at least implement a suboptimal solution. It is okay that yours is not optimal, people spent much effort to polish their solutions to optimize them.
3. Once you have a suboptimal solution, you may head over to the top solutions to learn what you can improve and any alternative methods to solve the same problem.

4. Try to thoroughly understand the thought process and implement the optimal solutions based on your understanding without looking at any hints.
5. Once you are comfortable with seeing through the problem patterns, it is time for the grand challenges.

Hard Problems

Hard problems are bar-raisers. They are intended to be hard and make you struggle. Usually, 45 minutes are barely enough for you to come up with a working solution. What you need to learn is identifying the right directions to solve the problems more than just brute force.

Checkpoint 3: Graduation Check

Hard problems usually have constraints that make the typical tricks not applicable. If you are comfortable with improving existing tricks to solve those problems more than brute force, you are good to go. The time limit is not that important here, you need to learn how to bridge the gap between typical tricks and those constraints.

Study Guide

1. Solving the problem is more important than finding the optimal solution. Your first task is to at least come up with a brute force solution. Dropping the time and/or space constraints usually help you identify one.
2. Identify what parts of your solution can be optimized to satisfy those constraints. This has been covered by many books and articles such as the BUD approach from CTCI so I would not go into details.
3. If you struggle to improve your solution, time to head to the top solutions. Understanding the thought process is critical here. You need to learn what are the right data structure and algorithms to use and how those solutions handle the corner cases.
4. Once you are comfortable with the stress from the hard problems, try to solve other hard problems with suboptimal solutions.

Thank you for reading. Hope you find this guide helpful. All critics and suggestions are welcome.